Ateliers Maths et Informatique, TP4

2 avril 2000

1 Introduction

Les algorithmes génétiques sont des algorithmes stochastiques d'optimisation conçu pour résoudre des problèmes complexes typiquement NP-complet. Ils s'inspirent des mécanismes de la génétique qui régissent l'Evolution au sens de Darwin.

Une population initiale de solutions potentielles au problème étudié, est générée aléatoirement. On la fait évoluer en lui appliquant successivement des opérateurs "génétiques": mutation, crossover (aussi appelé recombinaison) et sélection. De "meilleures" solutions émergent au cours de l'évolution, la qualité d'une solution étant évaluée grâce à une fonction appelée "fitness". Le problème original se réduit dès lors à trouver un optimum global de cette fonction. Noter que le succès de l'algorithme n'est en général pas garanti.

Algorithme génétique classique: pseudo-code

generation $= 0$
Initialize Population
while **not** *termination condition* **do**
    generation $=$ *generation* $+ 1$
    Compute Fitness
    Mutation($pMut$)
    Crossover($pCross$)
    Selection
end while

2 Problème du MAXONE

On considère le problème d'école du MAXONE qui consiste à maximiser le nombre de 1 dans une chaîne de bits de longueur fixée. Il s'agit d'un problème analogue à la maximisation du nombre de réponses justes à un ensemble de questions dont on ignore à priori les réponses. On propose d'implémenter avec Matlab un algorithme génétique classique qui résoud le problème du MAXONE. On fait donc évoluer une population de chaînes de bits à l'aide des opérations de mutation, crossover et sélection décrites ci-dessous.

Individu: une chaîne de bits de longueur `INDIV_LENGTH` sera représentée en écriture décimale, i.e. par un entier $s \in \{0, 1, \ldots, 2^{\texttt{INDIV\_LENGTH}} - 1\}$.

Population: une population de `popSize` individus sera représentée par un vecteur $(s_1, \ldots, s_{\texttt{popSize}})$ de `popSize` d'entiers dans $\{0, 1, \ldots, 2^{\texttt{INDIV\_LENGTH}} - 1\}$; la population initiale est choisie aléatoirement.

Fitness: la fitness $f(s)$ d'un individu $s \in \{0, 1, \ldots, 2^{\texttt{INDIV\_LENGTH}} - 1\}$ sera le nombre de bits à 1 dans sa représentation binaire.

Mutation: on flippe chaque bit d'un individu dans sa représentation binaire avec une faible probabilité `pMut`, chaque individu de la population mute indépendemment des autres.

Crossover: on constitue des paires d'individus de la forme $(2i-1, 2i)$ avec $i = 1, \ldots, \texttt{popSize}/2$ dans toute la population et chaque paire a une probabilité `pCross` de subir le crossover; le crossover d'une paire d'individu s'effectue en considérant leur représentation binaire et en choisissant aléatoirement un bit (crossover point) à partir duquel on recopiera les bits du deuxième individu dans le premier et vice-versa.

Sélection: étant donné une population $(s_1, \ldots, s_{\texttt{popSize}})$, on constitue un vecteur de probabilité $(p_1, \ldots, p_{\texttt{popSize}})$ avec

$$p_i = \frac{f(s_i)}{\sum_{i=1}^{\texttt{popSize}} f(s_i)}$$

à partir de la fitness de chacun des individus; on effectue `popSize` tirage avec remise dans la population $(s_1, \ldots, s_{\texttt{popSize}})$ avec les probabilités p_i pour obtenir une nouvelle population.

3 Travail à effectuer

Les 3 étapes successives de mutation, crossover et sélection donnent lieu à une génération de l'algorithme génétique. Voici le code Matlab du programme principal dont vous devrez écrire vous-même le code des fonctions `mutate, crossover` et `fitness`. Exécutez ce programme en faisant varier les paramètres et observer le comportement. Comparez les comportements avec et sans crossover ainsi qu'avec et sans mutation.

```
%clears the workspace from local and global variables
clear
clear global INDIV_LENGTH

%declaration of a global variable
global INDIV_LENGTH

%parameters of the run
INDIV_LENGTH=15
popSize=20
pMut=0.005
pCross=0.8
nbGen=1000

%initialization
population=random('Discrete Uniform',2^INDIV_LENGTH,popSize,1)-1;

%evolution
for i=1:nbGen
    %show generation
    i

    %fitness
    for j=1:popSize
        f(j)=fitness(population(j));
    end

    %mutation
    for j=1:popSize
        if rand<pMut
```

```
            population(j)=mutate(population(j),pMut);
            f(j)=fitness(population(j));
        end
    end

    %crossover
    for j=1:fix(popSize/2)
        if rand<pCross
            [population(2*j-1),population(2*j)]=crossover(population(2*j-1),population(2*j));
            f(2*j-1)=fitness(population(2*j-1));
            f(2*j)=fitness(population(2*j));
        end
    end

    %selection
    tempPop=population;
    tempFit=f;
    prob=cumsum(f)/sum(f);
    for j=1:popSize
        index=find(rand<=prob);
        population(j)=tempPop(index(1));
        f(j)=tempFit(index(1));
    end

    %data
    aveFit(i)=mean(f);
    maxFit(i)=max(f);
    maxIndex=find(f==maxFit(i));
    best(i)=population(maxIndex(1));

end

for i=1:2^INDIV_LENGTH
    totFit(i)=fitness(i-1);
end

%affichage
x(1:2^INDIV_LENGTH)=nbGen;
y(1:nbGen)=2^INDIV_LENGTH-1;
z(1:nbGen)=0;
plot3(x,(0:2^INDIV_LENGTH-1),totFit,'-rs',(1:nbGen),y,maxFit,'-gx',(1:nbGen),best,z,'-bd')
xlim([1,nbGen])
ylim([0,2^INDIV_LENGTH-1])
zlim([0,INDIV_LENGTH])
grid on
xlabel('generation')
ylabel('best individual')
zlabel('fitness')
text(nbGen,2^(INDIV_LENGTH-1),INDIV_LENGTH,'fitness landscape')
```